Platypus – A Multilingual Question Answering Platform for Wikidata

Thomas Pellissier Tanon, Marcos Dias de Assuncao, Eddy Caron, Fabian Suchanek

Platypus – A Multilingual Question Answering Platform for Wikidata

Technical report

Thomas Pellissier Tanon[1,2], Marcos Dias de Assunção[1], Eddy Caron[1], and Fabian M. Suchanek[2]

[1]Université de Lyon, ENS de Lyon, Inria, CNRS, Université Claude-Bernard Lyon 1, LIP
[2]LTCI, Télécom ParisTech

In this paper we present Platypus, a natural language question answering system. Our objective is to provide the research community with a production-ready multilingual question answering platform that targets Wikidata, the largest general-purpose knowledge base on the Semantic Web. Our platform can answer complex queries in several languages, using hybrid grammatical and template based techniques.

1 Introduction

Recent years have seen the rise of systems that can answer natural language questions such as "Who is the president of the United States?". These systems usually rely on *knowledge bases*. A knowledge base is a large, structured repository of machine-readable facts – usually encoded in the RDF model. Several large knowledge bases are freely available, including Wikidata [36], YAGO [32], and DBpedia [3]. Numerous question answering systems have been build on top of these knowledge bases [9, 29, 4, 34, 38, 18, 7, 11], and these systems perform well on benchmarks such as QALD [21], WebQuestions [4], and SimpleQuestions [6]. However, considerable work remains to be done to create systems that are user-friendly, multilingual and easy to extend. With these goals in mind we created Platypus, a multilingual question answering platform for Wikidata.

Platypus differs from existing systems in two main aspects. First, it explicitly targets Wikidata, which is the largest of the knowledge bases mentioned above. Second, it supports multiple natural languages with minimal adjustments.

Our method combines two approaches. The first one uses a set of transformation rules based on sentence grammars. This approach does not require training data, and it can be

adapted easily to different languages. The second approach is based on query templates and slot filling. It does require training data, but it can work even on questions with very little grammatical structure. Our system is available online as an API[1], as a Web interface[2], and as a Twitter bot[3].

This paper is structured as follows. Section 2 discusses related work. Section 3 presents our system, with Section 4 describing its implementation. Section 5 presents our experiments, followed by concluding remarks in Section 6.

2 Related Work

Question answering (QA) is an active domain of research since early 1970's.

One group of approaches is based on the grammatical structure [23] of the questions. Several works have augmented grammatical structures with semantics in several languages. Some of the approaches [1] focus on semantic role labeling, whereas others build logical representations from sentences [35, 12], and again others improve universal dependency trees with semantic relations [37]. Two QA systems work with this type of structures. One of them [22] uses the formal grammar of the target language. This approach requires large formal grammars to be built for each language. Our work, in contrast, relies solely on semantic parsing models, and does not need expensive adaptations for different languages. Furthermore, the work of [22] targets only domain-specific knowledge. Our work, in contrast can work on broad knowledge bases like Wikidata. The other QA approach is based on a multilingual semantic parser on top of universal dependencies [29]. Our work differs from this approach in two aspects. First, we provide a working implementation on Wikidata. Second, our logical representation matches directly the target knowledge base, so that we do not need an additional matching step. Thanks to this, our system can work with specific output types such as strings, dates, numbers, and quantities.

Another class of QA systems uses a logical representation based on the λ-calculus [40, 4]. Our approach also uses such a representation, but we show how to extend it to several languages.

Another group of QA approaches relies on machine learning in order to build a knowledge base query directly from the question. One approach [34] builds query templates based on the part-of-speech tree, and then instantiates them by a mapping to the knowledge base. Other work [38] focuses on the disambiguation process. Again other work [31] first performs an entity linking phase before building a SPARQL query, using a convolutional neural network. The recent trend is using end-to-end machine learning approaches, especially when targeting the SimpleQuestions dataset. The work by [6] introduces an approach based on memory networks. This work has been refined by [14] to use specific characters instead of words as input. The approach of [39] uses attentive convolutional networks, and [33] uses simple recurrent networks. There are some new attempts to

[1] `https://qa.askplatyp.us`
[2] `https://askplatyp.us`
[3] `https://twitter.com/askplatypus`

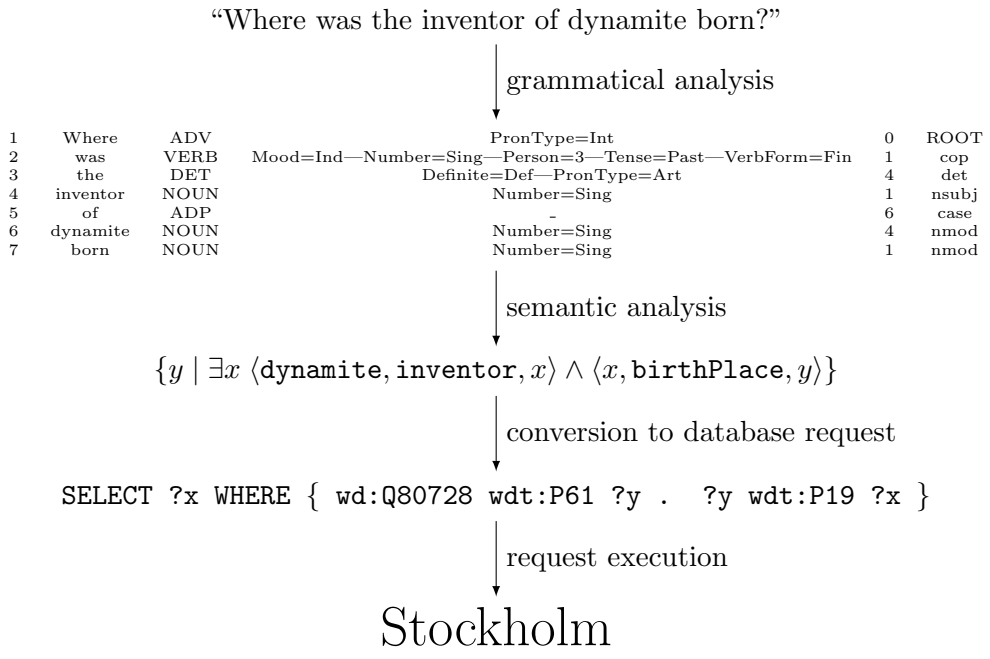


Figure 1: Pipeline execution using the grammatical analyzer

use these "machine learning" techniques for multilingual systems [13, 11]. Compared to these works, Platypus provides a similar end-to-end learning approach based on templates, but also a grammar based approach that can questions for which there is no template. Furthermore, our work provides easy support for multiple languages.

3 Platypus System

Our Platypus system takes as input a natural language question, and produces as output a set of answers from the Wikidata knowledge base. The system works in three steps. In the first step, the *analyzer* converts the natural language question into one or several internal *logical representations*. In the second step, the logical representations are ranked according to their likelihood of being the correct interpretation of the question. In the last step, the representations are converted into SPARQL, and executed one after the other on Wikidata, until one of them yields an answer. This modular architecture is designed to allow for different analyzers and different data sources. Indeed, we provide two different analyzers: a grammatical analyzer and a template-based one. Figure 1 shows the process with the grammatical analyzer.

Section 3.1 describes the logical representation that Platypus uses. Section 3.2 introduces the grammatical analyzer, whereas Section 3.3 describes the template-based one. Section 3.4 details the ranking of the presentations.

3.1 Logical Representation

Our question analyzers do not directly produce a well-known query language such as SQL or SPARQL [28], but rather a custom logical representation. The advantage of this approach is that it allows the composition of partial representations, which is a very useful feature for composition-based analyzers like the grammatical one. For instance, one could give the representation $\{x \mid \langle \texttt{dynamite}, \texttt{inventor}, x\rangle\}$ for "the inventor of dynamite" and $\{y \mid \langle x, \texttt{birthPlace}, y\rangle\}$ for "Where was X born?"; composing the two thus gives the representation $\{y \mid \exists x \ \langle \texttt{dynamite}, \texttt{inventor}, x\rangle \wedge \langle x, \texttt{birthPlace}, y\rangle\}$ for "Where was the inventor of dynamite born?".

Our logical representation is inspired by dependency-based compositional semantics [20, 19]. To define our representation formally, we assume a fixed set of *variables* $\mathcal{V}$. We also assume a fixed set of *entities* $\mathcal{T}$, which contains, e.g., the person Barack Obama or the integer 2. We will denote by $\mathcal{P} \subset \mathcal{T}$ the set of properties (such as "was born in"). We denote by $\mathcal{K} \subset \mathcal{T} \times \mathcal{P} \times \mathcal{T}$ our knowledge base. We use the set $\mathcal{B} = \{\texttt{true}, \texttt{false}\}$ for the two booleans. We will write $\phi[t/v]$ to mean the logical formula ϕ where all free occurrences of the variable v have been replaced by the expression t.

We will now define the space of logical representations $\mathcal{L}$, and the evaluation function $[\![]\!]$. The evaluation function takes as input an element of $\mathcal{L}$, and returns either a boolean value, or an entity from $\mathcal{T}$, or a tuple of entities. We denote with $\mathcal{L}_\mathcal{B} \subset \mathcal{L}$ the set of elements on which $[\![]\!]$ returns a boolean, and with $\mathcal{L}_\mathcal{T} \subset \mathcal{L}$ the set on which $[\![]\!]$ returns an entity. The space $\mathcal{L}$ and the function $[\![]\!]$ are defined recursively as follows:

Simple terms: For all $t \in \mathcal{T}$, we have $t \in \mathcal{L}_\mathcal{T}$. We define $[\![t]\!] := t$. For example, the person `Emmanuel Macron` is in $\mathcal{T}$, and $[\![\texttt{Emmanuel Macron}]\!] = \texttt{Emmanuel Macron}$.

Triples: For all $s, o \in \mathcal{L}_\mathcal{T}$ and all $p \in \mathcal{P}$, we have $\langle s, p, o\rangle \in \mathcal{L}_\mathcal{B}$. We define $[\![\langle s, p, o\rangle]\!] := [\langle [\![s]\!], p, [\![o]\!]\rangle \in \mathcal{K}]$. For example, the triple $\langle \texttt{France}, \texttt{president}, \texttt{Emmanuel Macron}\rangle$ is in $\mathcal{L}_\mathcal{B}$, and its evaluation is `true` if Macron is the president of France in $\mathcal{K}$.

Introduction of variables: For all variables $v \in \mathcal{V}$, and for all representations $\phi \in \mathcal{L}_\mathcal{B}$, the formula $\exists v.\phi$ is in $\mathcal{L}_\mathcal{B}$. We define $[\![\exists v.\phi]\!] := \texttt{true}$ iff there exists $t \in \mathcal{T}$ such that $[\![\phi[t/v]]\!] = \texttt{true}$. For example $\exists o.\langle \texttt{France}, \texttt{president}, o\rangle$ is a logical representation that states that there is a president o in France, and its evaluation will be true if there is indeed such an o in $\mathcal{K}$.

Sets defined by a boolean formula: For all $\phi \in \mathcal{L}_\mathcal{B}$ with free variables $v_1, \ldots, v_n \in \mathcal{V}$, we have $\{(v_1, \ldots, v_n) \mid \phi\} \in \mathcal{L}$. We define $[\![\{(v_1, \ldots, v_n) \mid \phi\}]\!] := \{(t_1, \ldots, t_n) \mid [\![\phi[t_1/v_1] \cdots [t_n/v_n]]\!]\}$. For example, $\{x \mid \langle x, \texttt{work in}, \texttt{Paris}\rangle\}$ is the set of all people working in Paris.

Elements in a set: For all $\{(v_1^b, \ldots, v_n^b) \mid \phi\} \in \mathcal{L}$ and $v_1^f, \ldots, v_n^f \in \mathcal{V}$, we have $((v_1^f, \ldots, v_n^f) \in \{(v_1^b, \ldots, v_n^b) \mid \phi\}) \in \mathcal{L}_\mathcal{T}$. We define $[\![(v_1^f, \ldots, v_n^f) \in \{(v_1^b, \ldots, v_n^b) \mid \phi\}]\!] = [\![\phi[v_1^f/v_1^b] \cdots [f_n^f/v_n^b]]\!]$. For example, we could state that Emmanuel Macron works in Paris with $\texttt{Emmanuel Macron} \in \{x \mid \langle x, \texttt{work in}, \texttt{Paris}\rangle\}$.

Conjunction: For all $\phi, \psi \in \mathcal{L}_\mathcal{B}$, we have $\phi \wedge \psi \in \mathcal{L}_\mathcal{B}$. We define $[\![\phi \wedge \psi]\!] := [[\![\phi]\!] \wedge [\![\psi]\!]]$. For example, we can represent the birth place of the president of France as $\{l \mid \exists p. \langle \texttt{France}, \texttt{president}, p\rangle \wedge \langle p, \texttt{birth place}, l\rangle\}$

Disjunction: For all $\phi, \psi \in \mathcal{L}_\mathcal{B}$, we have $\phi \vee \psi \in \mathcal{L}_\mathcal{B}$. We define $[\![\phi \vee \psi]\!] := [[\![\phi]\!] \vee [\![\psi]\!]]$. For example, we can represent the set of presidents of France and the United States as $\{p \mid \langle \texttt{France}, \texttt{president}, p\rangle \vee \langle \texttt{United States}, \texttt{president}, p\rangle\}$.

Negation: For all $\phi \in \mathcal{L}_\mathcal{B}$, we have $\neg\phi \in \mathcal{L}_\mathcal{B}$. We define $[\![\neg\phi]\!] := [\neg[\![\phi]\!]]$. For instance, the president of France was not born in Germany: $\exists p. \langle \texttt{France}, \texttt{president}, p\rangle \wedge \neg\langle p, \texttt{born in}, \texttt{Germany}\rangle$.

Comparisons: For all $a, b \in \mathcal{L}_\mathcal{T}$, we have $[a = b] \in \mathcal{L}_\mathcal{B}$. We define $[\![[a = b]]\!] := [[\![a]\!] = [\![b]\!]]$. The same goes for the operators $\neq, \geq, \leq, <, >$. For example $[1 > 2]$ is a valid logical representation, and its evaluation is true.

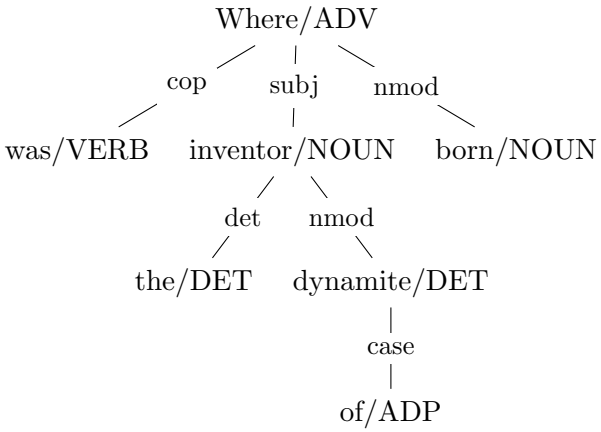
Arithmetic operations: For all $t_1, t_2 \in \mathcal{L}_\mathcal{T}$, we have $t_1 + t_2 \in \mathcal{L}_\mathcal{T}$. We define $[\![t_1 + t_2]\!] := [\![t_1]\!] + [\![t_2]\!]$. The same goes for the other arithmetic operators $-, *, /$. For example, the double of the population of Lyon can be written as $\{v \mid [v = 2 * p] \wedge \langle \texttt{Lyon}, \texttt{population}, p\rangle\}$.

Our system can simplify these expressions in some cases. For example:

- If $dom(p)$ and $range(p)$ are the domain and range(s) of the property $p \in \mathcal{P}$, then for all $s, o \in \mathcal{T}$ if $s \notin dom(p)$ or $o \notin range(p)$ then $[\![\langle s, p, o\rangle]\!] = false$.
- The expression $(v_1^f, \ldots, v_n^f) \in \{(v_1^b, \ldots, v_n^b) \mid \phi\}$ can be simplified to $\phi[v_1^f/v_1^b] \cdots [f_n^f/v_n^b]$.

3.2 Analyzer 1: Grammatical analyzer

The main analyzer of Platypus is the grammatical analyzer. It takes as input a natural language question, and translates it into a logical representation. For this purpose, it first parses the question with standard tools, yielding a dependency path. We use the CoNLL-U format [24] based on the Universal Dependencies parts-of-speech (POS) and dependency tag sets. For example the sentence "Where was the inventor of dynamite born?" becomes:



From this tree, we want to build the following logical representation:

$$\{y \mid \exists x \ \langle \texttt{dynamite}, \texttt{inventor}, x\rangle \wedge \langle x, \texttt{birthPlace}, y\rangle\}$$

This transformation is achieved by rules that use two functions, $parse_t$ and $parse_r$. $parse_t$ is the main parsing function, which parses a dependency tree in order to return its logical representation. For example, for the parse tree corresponding to the sentence "the inventor of dynamite", it returns the logical representation $\{x \mid \langle \texttt{dynamite}, \texttt{inventor}, x\rangle\}$.

$parse_r$ is a utility function that parses properties. For example, for the parse tree corresponding to the words "birth place", the function returns the logical representation $\{(x, y) \mid \langle x, \texttt{birthPlace}, y\rangle\}$.
We give here some examples of transformation rules:

- If $x \in \mathcal{T}$ is an entity that could be represented by the lexeme X, then $parse_t(X) = \{p \mid [p = x]\}$. For example, "Paris" could represent the capital of France, and hence $parse_t(Paris) = \{p \mid [p = \texttt{Paris}]\}$.
- If $p \in \mathcal{P}$ is a property that could be represented by lexeme R, then $parse_p(R) = \{(s, o) \mid \langle s, r, o\rangle\}$. For example, $parse_p(author) = \texttt{http://schema.org/author}$.
- $parse_t\left(\begin{array}{c}\texttt{Where}\\ \vert\ \text{nsubj}\\ \text{X}\end{array}\right) = \left\{p \mid \exists x.\, x \in parse_t(X) \wedge \langle x, \texttt{located in}, p\rangle\right\}$
- $parse_t\left(\begin{array}{ccc}&\texttt{P}&\\ \text{case}/&&\backslash\ \text{nmod}\\ \texttt{of}&&\text{S}\end{array}\right) = \left\{o \mid \exists s.\, s \in parse_t(S) \wedge (s, o) \in parse_r(P)\right\}$

We give here some examples of simple sentence analyses:

$$parse_t\left(\begin{array}{ccc} & \text{Where} & \\ \text{cop}\diagup & & \diagdown\text{nsubj} \\ \text{is} & & \text{Paris} \end{array}\right) = \left\{ p \,\middle|\, \exists x. \begin{array}{r} x \in parse_t(Paris) \wedge \\ \langle x, \texttt{located in}, p\rangle \end{array} \right\}$$

$$= \{p \mid \langle \texttt{Paris}, \texttt{located in}, p\rangle\}$$

$$parse_t\left(\begin{array}{ccc} & \text{Father} & \\ \text{case}\diagup & & \diagdown\text{nmod} \\ \text{of} & & \text{Obama} \end{array}\right) = \left\{ o \,\middle|\, \exists s. \begin{array}{r} s \in parse_t(Obama) \wedge \\ (s, o) \in parse_r(father) \end{array} \right\}$$

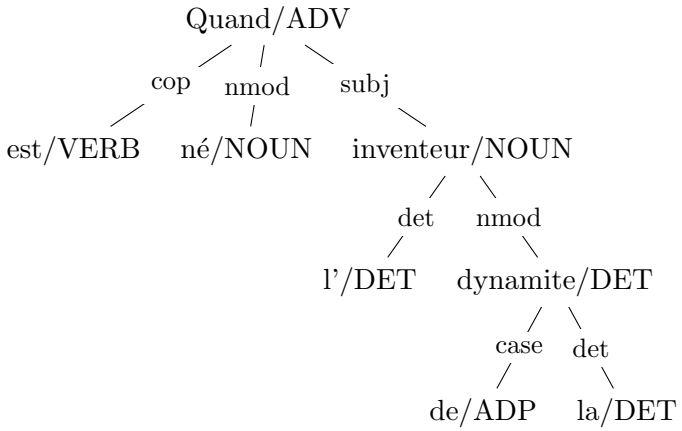
$$= \{o \mid \langle \texttt{Obama}, \texttt{father}, o\rangle\}$$

When several rules can be applied, the analyzer returns several results. Hence, $parse_t$ does not return a single logical representation but a set of possible representations. We will discuss in Section 3.4 how to filter out the wrong representations.

3.2.1 Multilingual Parsing.

Our rules depend only on the set of POS tags and the set of dependency tags – and not on the input language. Both tag sets are language independent. The only rules that have to be adapted to the input language are those that contain relation words such as "of" and "in", or question markers such as "when" or "where". These words can either express a relation (as in "Where is London?") or modify a relation (as in "Where was Alfred Nobel born?"). In the latter case, we have to combine both "where" and "born" to find the relation "birth place".

For example, the parse tree for the translation of "Where was the inventor of dynamite born?" in French ("Quand est né l'inventeur de la dynamite?") is:



We can apply the same analysis to the French question "Où est Paris" as to its English translation "Where is Paris":

Table 1: Examples of annotated training set for the template analyzer.

logical representation template	training samples
$\{o \mid \langle \text{s}, \texttt{birth date}, o\rangle\}$	When was George^s Washington^s born? birth year Obama^s
$\{o \mid \langle \text{s}, \text{p}, o\rangle\}$	Paris^s population^p place^p of^p birth^p of Albert^s Einstein^s What is the nickname^p of New^s York^s ?
$\{v \mid [v = \text{m} * o] \wedge \langle \text{s}, \text{p}, o\rangle\}$	Twice^m the population^p of France^o half^m Bill^o Gates^o wealth^p

$$parse_t\left(\begin{array}{ccc} & \text{Où} & \\ \text{cop} & / & \backslash \ \text{nsubj} \\ \text{est} & & \text{Paris} \end{array}\right) = \{p | \exists x.\, x \in parse_t(\texttt{Paris}) \wedge \langle x, \texttt{located in}, p\rangle\}$$

$$= \{p | \langle \texttt{Paris}, \texttt{located in}, p\rangle\}$$

3.3 Analyzer 2: Template analyzer

The second Platypus analyzer is based on templates. A template is a logical representation with free variables, annotated with natural language questions. In these questions, the entities are tagged with the free variables of the template. Table 1 shows some examples.

We discuss in Section 5 how to obtain these templates from previous work. Our analyzer uses these templates in order to find the logical representation of a given natural language question. For this purpose, the analyzer first finds the template that best matches the question. This is done using a classifier. We encode the question by the average of the word embeddings of its words, and classify them with a linear support vector machine. After this, the analyzer fills the logical representation slots. We use conditional random fields [17] to recognize entities in the input sentence, and we match them with the knowledge base entities using the same techniques as in the grammatical analysis.

3.4 Ranking Logical Representations

As our analyzers may return several logical representations for the same input sentence, we must rank interpretations by likelihood. For example, we want to return first the birth date of George Washington the US president, and not of George Washington the inventor of an early version of instant coffee. For this purpose, we assume a ranking function on the entities. For Wikidata, the entity ranking is done by counting the number of languages in which the entity has a Wikipedia article. In the future, this

function could be adapted for different use cases. For example, a shopping assistant could use the distance between a shop and the user to do its ranking.

The ranking score of a logical representation is then computed as the maximum of the ranking score of each entity mentioned in the representation, minus a penalty for the complexity of the representation. The complexity of the representation is the number of constructors used in the representation (without set definition and conjunction operators). For example, the complexity of $\{v \mid [v = m * o] \wedge \langle s, \mathtt{p}, o\rangle\}$ is 2 because the expression contains one product and one triple. Penalizing complex representations is useful for dealing with cases such as the movie title "Who Framed Roger Rabbit". This movie title could be interpreted either simply as the movie title, or as an actual question asking for the person who framed the entity "Roger Rabbit".

After ranking the representations in this way, we execute them one after the other, and return the results of the first query whose results are not empty. This allows us to cope gracefully with wrong parsings or empty results.

Compared to [34], our ranking function is quite simple. This is because most of the wrong representations that our analyzers generate do not return any results on the knowledge base anyway. Therefore, we can stay with a very compact ranking function.

4 Implementation

Our approach is largely independent of the actual knowledge base. For example, for our evaluation, we have implemented the approach for both Wikidata [36] and Freebase. This is possible because the interaction with the knowledge base is abstracted into an interface. This interface provides methods 1) to retrieve entities and properties from a label and optionally a class, 2) to execute logical representation against the knowledge base and 3) to do entity formatting. For Wikidata, our implementation of the interface uses a specialized service to perform fast entity-lookup with support of edit distance and type constraints. We also developed a converter between our logical representation and the SPARQL query language[4]. In order not to overload the Wikidata SPARQL endpoint, Platypus has its own data storage. To keep our answers accurate, we perform a daily replication of Wikidata to include updates. Wikidata lends itself for Platypus for two reasons. First, Wikidata provides a large set of lexical representations for its properties, in numerous languages [16] (e.g., "was born in", "has the birthplace", and "est né à" for `bornIn`). Second, Wikidata is one of the largest general purpose knowledge bases on the Semantic Web – especially since Freebase was merged into Wikidata [26].

Our grammatical parsing is also abstracted into an interface. It takes as input a natural language question, and returns as output a Universal Dependencies parse. We implemented this interface for the parsers CoreNLP [8], SyntaxNet [2], and Spacy [15].

The grammatical analyzer is implemented in Python. It uses dictionaries of connection words (such as "in" or "from") and question words (such as "where" or "when"). We have developed dictionaries for English, French, which allows Platypus to answer questions in

[4] `http://www.w3.org/TR/sparql11-query/`

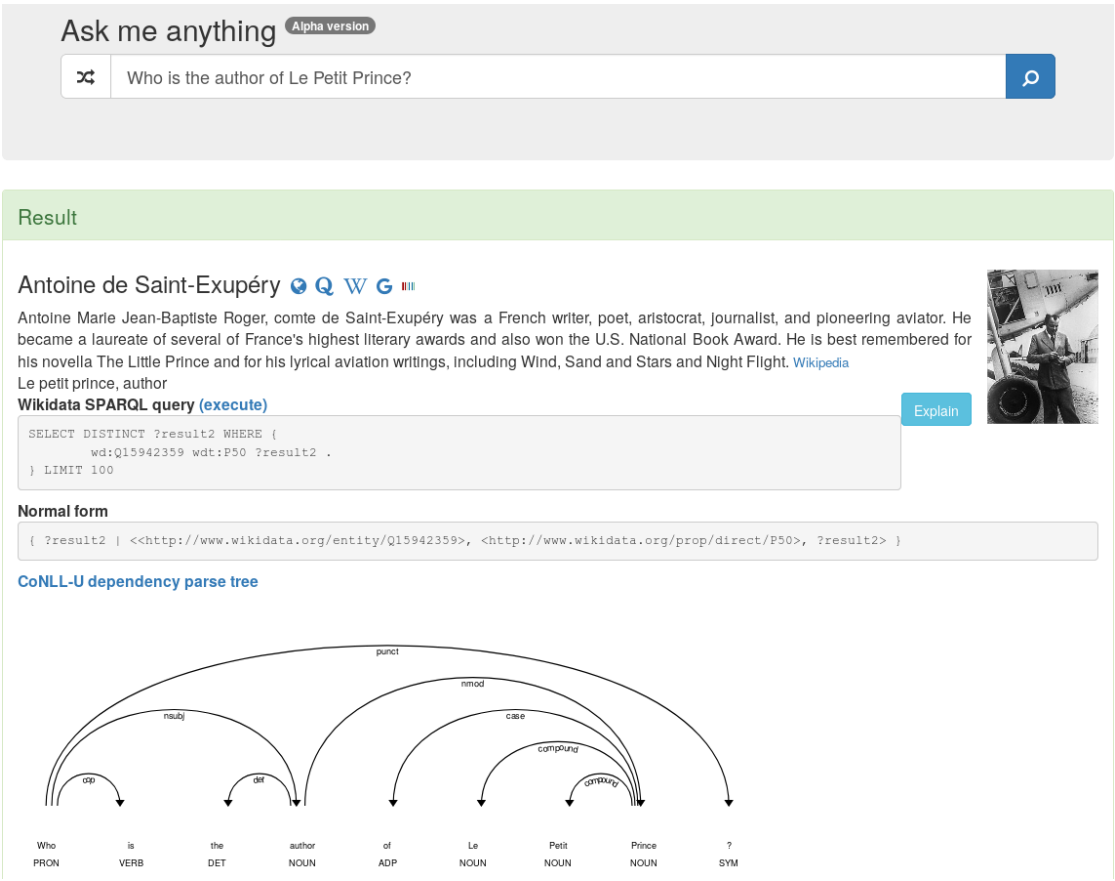


Figure 2: Platypus web user interface

these two languages. The support of Spanish and German is currently in development. Language detection is done using a Python port of [30].

The template analyzer is implemented using RasaNLU [5]. We used the Glove [27] word vectors trained on Common Crawl provided by Spacy and the RasaNLU entity extractor based on the CRFsuite library [25].

Our system can be accessed in three ways. First, there is a simple REST API that takes a question string as input, and returns a set of RDF graphs as result. Each graph is annotated with its score and the the logical representation used to retrieve it. This API is available at `https://qa.askplatyp.us`. The second access method is a Web interface, which is available at `https://askplatyp.us`. Finally, in order to experiment with different interactions, we also implemented a Twitter bot. The bot replies to tweets sent to this account. We take advantage of the mapping of 100k Twitter accounts to Wikidata items (as of January 2018) to mention the relevant Twitter accounts in replies. The bot is available at `https://twitter.com/askplatypus`.

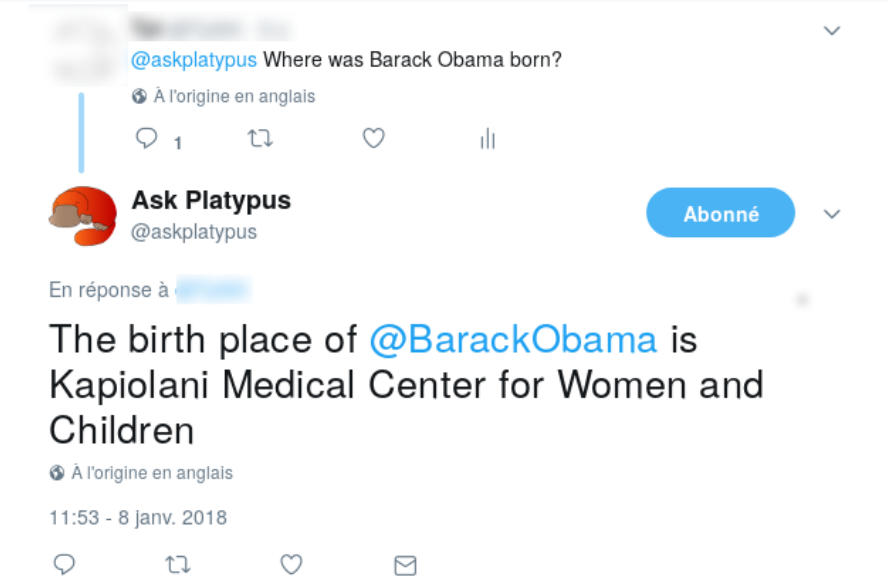


Figure 3: Platypus Twitter bot

5 Evaluation

For evaluating the template analyzer (see Section 3.3), we use the SimpleQuestions dataset [6]. It was introduced in 2015, and has been used to evaluate a significant number of question answering systems [14, 39, 33]. The dataset is split into training, validation, and test parts, with 75,910 questions in the training part, 10,845 in the validation part, and 21,687 in the test part. The dataset has been converted to Wikidata [10]. It provides 34,374 training questions, 4,867 validation questions, and 9,961 test questions. In both versions, each question is annotated with a knowledge base triple $\langle s, p, o \rangle$, so that `SELECT ?o WHERE` { s p `?o .` } is the SPARQL query that should be run in order to retrieve answer o to the query. For example, in the Freebase version of the datset, the question "When was Barack Obama born" is annotated with $\langle$`/m/02mjmr`, `/people/person/date_of_birth`, `"1961-08-04"`$\rangle$. Here, `/m/02mjmr` is the Freebase identifier for Obama.

We converted these datasets to our template analyzer format as follows: We first built one logical form template $\{o \mid \langle s, p, o \rangle\}$ for each relation p in the dataset. We joined to these patterns all the questions annotated by the given property. Then, we annotated the questions with the missing s slots. To do so, we retrieved all the labels and aliases of the expected value of slot s and we looked in the question for the longest occurrence of one of these labels or aliases, while allowing a short edit distance (2 at most) to take care of capitalization and pluralization differences. If we find such an occurrence, we tag it as the expected value of slot s, if not we prune out the question.

Table 2: Evaluation results on SimpleQuestions

system	precision (%)	recall (%)
MemNN [6]	63.9	100
Char-level CNN [14]	70.9	100
Attentive max-pooling [39]	76.4	100
RNN-QA [33]	88.3	100
Platypus (Freebase)	64.1	89.5
Platypus (Wikidata)	73.9	87.1

For both Freebase and Wikidata, we use the train and evaluation parts of this dataset to train the template analyzer, and we evaluated it on the test part. We then proceed to the same evaluation as [6]: we consider a prediction correct if the subject and the predicate are correctly extracted (i.e., if the logical representation is valid). In this way, our evaluation measures only the correctness of the interpretation of the question, and it does not depend on the completeness of the knowledge base.
Table 5 shows the evaluation results of Platypus on both Freebase and Wikidata. For comparison, we also show the performance of the state of the art systems on the Freebase version of SimpleQuestions. For systems that perform differently on different variants of Freebase, we give the best score obtained on either the FB2M or FB5M Freebase subsets.

Our results show that Platypus performs roughly comparably to the state of the art on Freebase. At the same time, it is the only one of the systems that can also work on Wikidata.

6 Conclusion

In this paper, we have introduced Platypus, a multilingual natural language question answering system for Wikidata. Platypus can work with both a grammatical analyzer and a template-based analyzer to parse natural language questions. These algorithms can be adapted easily to other languages. Platypus can be tried out online in English, German, French, and Spanish on our Web page `https://askplatyp.us`.

Acknowledgments

We thank the contributors of the first version of the Platypus project, namely Marc Chevalier, Raphaël Charrondière, Quentin Cormier, Tom Cornebize, Yassine Hamoudi, Valentin Lorentz.

This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon.

References

[1] Alan Akbik, Laura Chiticariu, Marina Danilevsky, Yunyao Li, Shivakumar Vaithyanathan, and Huaiyu Zhu. Generating high quality proposition banks for multilingual semantic role labeling. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL*, pages 397–407, 2015.

[2] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL*, 2016.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web, 6th International Semantic Web Conference, ISWC*, pages 722–735, 2007.

[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1533–1544, 2013.

[5] Tom Bocklisch, Joey Faulker, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management. *CoRR*, 2017.

[6] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, 2015.

[7] Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. Qakis: an open domain QA system based on relational patterns. In *Proceedings of the ISWC 2012 Posters & Demonstrations Track*, 2012.

[8] Danqi Chen and Christopher D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 740–750, 2014.

[9] Dennis Diefenbach, Vanessa Lopez, Kamal Singh, and Pierre Maret. Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information Systems*, Sep 2017.

[10] Dennis Diefenbach, Thomas Pellissier Tanon, Kamal Deep Singh, and Pierre Maret. Question answering benchmarks for wikidata. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference ISWC*, 2017.

[11] Dennis Diefenbach, Kamal Singh, and Pierre Maret. Wdaqua-core0: a question answering component for the research community. *ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7)*, 2017.

[12] Matt Gardner and Jayant Krishnamurthy. Open-vocabulary semantic parsing with both distributional statistics and formal knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3195–3201, 2017.

[13] Sherzod Hakimov, Soufian Jebbara, and Philipp Cimiano. AMUSE: multilingual semantic parsing for question answering over linked data. In *The Semantic Web, 16th International Semantic Web Conference, ISWC*, pages 329–346, 2017.

[14] Xiaodong He and David Golub. Character-level question answering with attention. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1598–1607, 2016.

[15] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1373–1378, 2015.

[16] Lucie-Aimée Kaffee, Alessandro Piscopo, Pavlos Vougiouklis, Elena Simperl, Leslie Carr, and Lydia Pintscher. A glimpse into babel: An analysis of multilinguality in wikidata. In *Proceedings of the 13th International Symposium on Open Collaboration, OpenSym*, pages 14:1–14:5, 2017.

[17] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning ICML*, pages 282–289, 2001.

[18] Jens Lehmann and Lorenz Bühmann. Autosparql: Let users query your knowledge base. In *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC*, pages 63–79, 2011.

[19] Percy Liang. Lambda dependency-based compositional semantics. 2013.

[20] Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.

[21] Vanessa López, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating question answering over linked data. *Journal of Web Semantics*, 21:3–13, 2013.

[22] Anca Marginean. Question answering over biomedical linked data with grammatical framework. *Semantic Web*, 8(4):565–580, 2017.

[23] Richard Montague. The proper treatment of quantification in ordinary english. In *Philosophy, language, and artificial intelligence*, pages 141–162. 1973.

[24] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC*, 2016.

[25] Naoaki Okazaki. Crfsuite: a fast implementation of conditional random fields. 2007.

[26] Thomas Pellissier Tanon, Denny Vrandecic, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. From freebase to wikidata: The great migration. In *Proceedings of the 25th International Conference on World Wide Web, WWW*, pages 1419–1428, 2016.

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1532–1543, 2014.

[28] Eric Prud'hommeaux, Andy Seaborne, et al. SPARQL query language for RDF. 2006.

[29] Siva Reddy, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 89–101, 2017.

[30] Nakatani Shuyo. Language detection library for java, 2010.

[31] Daniil Sorokin and Iryna Gurevych. End-to-end representation learning for question answering with weak supervision. *ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7)*, 2017.

[32] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW*, pages 697–706, 2007.

[33] Ferhan Türe and Oliver Jojic. No need to pay attention: Simple recurrent neural networks work! In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 2856–2862, 2017.

[34] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over RDF data. In *Proceedings of the 21st World Wide Web Conference WWW*, pages 639–648, 2012.

[35] Lucy Vanderwende, Arul Menezes, and Chris Quirk. An AMR parser for english, french, german, spanish and japanese and a new amr-annotated corpus. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 26–30, 2015.

[36] Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communication of the ACM*, 57(10):78–85, 2014.

[37] Aaron Steven White, Drew Reisinger, Keisuke Sakaguchi, Tim Vieira, Sheng Zhang, Rachel Rudinger, Kyle Rawlins, and Benjamin Van Durme. Universal decompositional semantics on universal dependencies. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1713–1723, 2016.

[38] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL*, pages 379–390, 2012.

[39] Wenpeng Yin, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze. Simple question answering by attentive convolutional neural network. In *Proceedings of the 26th International Conference on Computational Linguistics, COLING, Technical Papers*, pages 1746–1756, 2016.

[40] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI*, pages 1050–1055, 1996.